

© 2023 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works.

Navardi, Mozhgan, Edward Humes, Tejaswini Manjunath, and Tinoosh Mohsenin. "MetaE2RL: Toward Metareasoning for Energy-Efficient Multi-Goal Reinforcement Learning with Squeezed Edge YOLO." IEEE Micro, 2023, 1–9. <https://doi.org/10.1109/MM.2023.3318200>.

<https://doi.org/10.1109/MM.2023.3318200>

Access to this work was provided by the University of Maryland, Baltimore County (UMBC) ScholarWorks@UMBC digital repository on the Maryland Shared Open Access (MD-SOAR) platform.

Please provide feedback

Please support the ScholarWorks@UMBC repository by emailing scholarworks-group@umbc.edu and telling us what having access to this work means to you and why it's important to you. Thank you.

Article Type: Description (see Introduction for more detail)

MetaE2RL: Toward Metareasoning for Energy-Efficient Multi-Goal Reinforcement Learning with Squeezed Edge YOLO

Mozhgan Navardi, *Johns Hopkins University, Baltimore, MD, 21218, USA*

Edward Humes, *University of Maryland Baltimore County, Baltimore, MD, 21250, USA*

Tejaswini Manjunath, *University of Maryland Baltimore County, Baltimore, MD, 21250, USA*

Tinoosh Mohsenin, *Johns Hopkins University, Baltimore, MD, 21218, USA*

Abstract—Metareasoning shows promise in efficiently using the computational resources of tiny edge devices while performing highly computationally intensive Reinforcement Learning (RL) algorithms. This work proposes MetaE2RL: a hardware-aware framework that incorporates low-power pre-processing solutions and metareasoning to enable the deployment of multi-goal RL on tiny autonomous devices. For this aim, a meta-level is proposed to allocate resources efficiently in real-time by switching between models with different complexities. Moreover, Squeezed Edge YOLO is proposed for energy-efficient object detection in the pre-processing phase. For the experimental results, the proposed Squeezed Edge YOLO was deployed onboard a tiny drone named Crazyflie with GAP8 processor that includes 8 parallel RISC-V cluster cores. We compared latency and power consumption of Squeezed Edge YOLO and a lighter CNN-based model while deploying them separately onboard on GAP8. Experimental results show Squeezed Edge YOLO is 8x smaller than previous work and consumes 541 mW on GAP8 with inference latency of 130 ms.

With the increasing capabilities of autonomous systems, tiny Unmanned Aerial Vehicles (UAVs) or Unmanned Ground Vehicles (UGVs) can provide a low-cost solution for real-time situational awareness and decision-making support. These resource-constrained devices have been utilized pervasively in many indoor applications such as search and rescue, source seeking, and real-time monitoring, which can be too dangerous and unsafe for humans [1]. For this purpose, small and agile drones are required to be able to autonomously navigate in GPS-denied environments. Artificial Intelligence (AI) in the form of Reinforcement Learning (RL) and object detection models such as YOLO have shown promise in enabling autonomous drone navigation [2], [3]. Moreover, autonomous systems are being

equipped with an increasing variety sensors such as cameras and LIDAR which can provide complementary sensing information. Consequently, this enables robust perception performance and higher reliability in navigation and accomplishing mission objectives while collaborating with other devices and humans. However, the limited power and computational capacity in autonomous edge systems necessitates energy-efficient deployment of RL and YOLO on such devices. Existing work [4] addresses this by introducing an efficient language-guided RL framework and an optimized object detection model named Edge YOLO. However, to the best of our knowledge, there is no tiny Machine Learning (tinyML) hardware solution leveraging multi-goal RL and YOLO-based models for resource-constraint autonomous systems.

Resource-constrained devices like tiny UAVs have limited sources of power and computation capacity. Therefore, for edge computing, object detection models must be optimized with regard to the number

XXXX-XXX © 2021 IEEE
Digital Object Identifier 10.1109/XXX.0000.00000000

of parameters and computations. These requirements have led to tinyMLs emergence for bringing complex neural networks to low-power, resource-constrained devices. TinyML is able to significantly reduce both energy and latency in comparison with cloud-based approaches, as all processing is performed on the edge. However, model optimization techniques such as pruning and quantization reduce the model size and can lead to a potentially unacceptable accuracy drop. As the application space is continuously changing for edge devices it is vital to prevent misprediction.

Meta-reasoning occurs whenever an algorithm adjusts the agent's decision-making process based on its current situation [5], [6]. As a result the agent, in this case, a drone, can dynamically switch between different algorithms while considering its power consumption, latency constraints, and model accuracy metrics. Meta-reasoning as a higher-level unit monitors the environment and provides the best algorithm for the current situation. In this paper, we proposed MetaE2RL, a meta-reasoning approach for energy-efficient reinforcement learning and YOLO object detection in autonomous systems that applies a meta-reasoning policy to improve latency and energy efficiency. The main contribution of this work is summarized as follows:

- An energy-efficient autonomous navigation framework is presented in this work which uses multi-goal RL and Squeezed Edge YOLO models to train the agent to reach the goal.
- A Squeezed Edge YOLO as a tinyML is proposed which is optimized for tiny edge devices.
- A meta-reasoning policy is proposed to switch between different algorithms in real-time based on environmental changes to improve power consumption and latency.
- Power and latency analysis on GAP8 processor as a result of the memory hierarchy and onboard data transmission overhead.

Preliminaries

Reinforcement Learning (RL)

Hierarchical goal-based tasks, which involve accomplishing a series of sub-goals to achieve a larger goal, can be modeled as a Markov decision process (MDP). An MDP consists of a state-space S , an action space A , a reward function $r: S \times A \rightarrow R$, an initial state distribution s_0 , and a transition probability $p(s_{t+1}|s_t, a_t)$ that defines the probability of transitioning from state s_t and action a_t at time t to next state s_{t+1} . To model

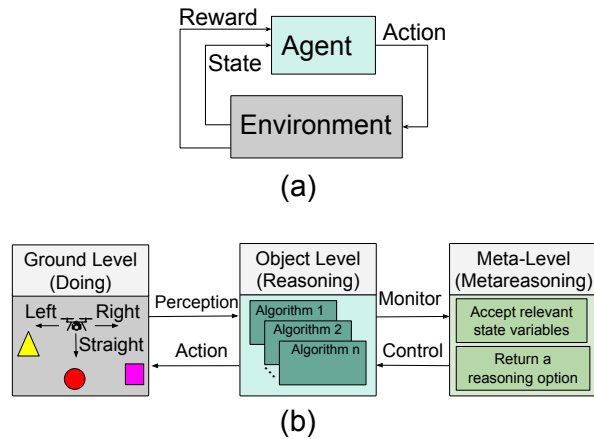


FIGURE 1. (a) Reinforcement Learning (RL) high-level diagram showing how the agent interacts with the environment to maximize the reward, (b) Metareasoning approach includes three levels: Ground level, Object level, and Meta-level.

hierarchical goal-based tasks in an MDP, the task can be decomposed into a hierarchy of sub-goals, where each sub-goal is itself a goal-oriented task [7]. The MDP process, as shown in Figure 1 (a), illustrates how an agent interacts with the environment to receive rewards.

Metareasoning

Metareasoning, deciding how to allocate computational resources, contains within it the solution to the problem of how to efficiently deploy computational resources. Metareasoning is reasoning about reasoning, which means making intelligent decisions about how to think [8]. Figure 1 (b) illustrates a high-level diagram of the metareasoning which includes: ground level, object level, and meta-level. The first two levels, can be considered as the environment and agent units in RL. Meta-level monitors the object level to control the agent in the online phase based on the real-time changes.

Metareasoning for Energy-Efficient Autonomous Navigation in Tiny Edge Devices

Motivation

Multi-goal RL approaches are better than training a policy per goal due to knowledge transfer through off-policy learning. Concatenating state and goal data as a unique input is not the best solution since it only covers a small subset of mappings between states and goals [9], [10]. However, with reduced state space,

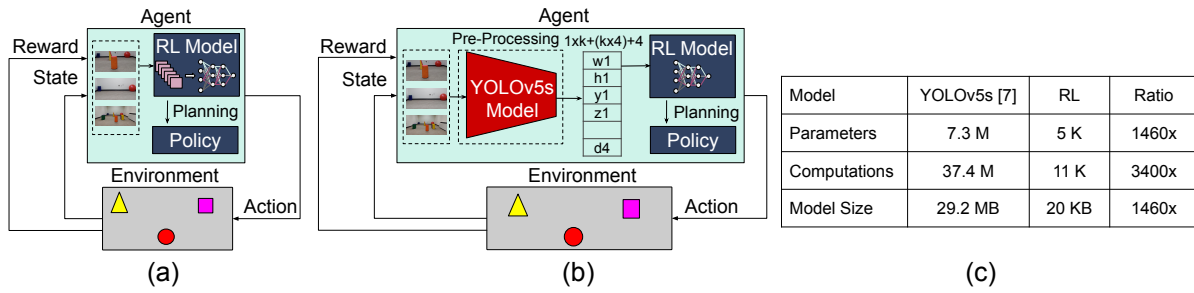


FIGURE 2. (a) Block-level diagram of Deep Reinforcement Learning (DRL), (b) ReProHRL [7] block-diagram, considered as a baseline in this paper, which adds YOLOv5s model as a pre-processing step to simplify RL model input to a 1×19 vector for three goals instead of processing the whole of the image. However, it adds highly intensive computational overhead and prevents applying this approach for edge computing in tiny edge device applications. (c) The bottleneck of adding a pre-processing module like YOLO before RL. We compare the number of parameters, computation, and model size of the RL and YOLOv5s models.

mappings become easier. The proposed ReProHRL in [7] only adds a small vector to the state space for each object, making it easily transferable to other environments. This allows for the use of the naive approach of Universal Value Function Approximators (UVFA).

Figure 2 illustrates block-level representation of RL agent and ReProHRL [7]. Figure 2 (b) illustrates the architecture of ReProHRL which in the environment unit agent search for a sub-goal and send feedback to the agent unit. In the agent unit, authors add a pre-processing unit to extract Bounding Box (BBox) and concatenate it with the given sub-goal in higher level policy unit and Lidar sensor outputs. The concatenated data, a $1 \times k + (4 \times k) + 4$ vector where k is the number of goals, is sent to the RL model unit and policy unit which processes to generate the actions. However, they add a YOLO model to their architecture which is highly intensive computing and prevents deploying this approach to tiny edge devices. Figure 2 (c) shows YOLOv5s model size is 1460x bigger than RL model. Therefore, it is vital to propose an approach to address this challenge. As the Bounding Box (BBox) given by the YOLO model is a key factor in this architecture, we cannot completely replace the YOLO model with lighter CNN-based models, as they can only classify objects. Thus, in this work a Squeezed Edge YOLO is proposed to replace the YOLOv5s model in the ReProHRL [7] architecture. Moreover, a CNN-based lighter model is added to the architecture as we do not need the YOLO model BBox output all the time. Finally, a metareasoning policy is proposed to switch between two models to enable energy-efficient edge computing.

Proposed Approach

Figure 3 depicts a system overview of the proposed approach named **MetaE2RL**, **Meta**reasoning for **E**nergy **E**fficiency of multi-goal **RL**, for tiny edge devices. MetaE2RL trains agent to reach multi goals in the environment. We add a higher level to the RL architecture as the meta-level which monitors the environmental changes to help the agent for decision-making in real-time. As shown in Figure 3, MetaE2RL includes a ground level, an object level, and a meta-level. At the ground level, the agent takes the given action (straight, left, and right) from the object level and sends the reward and state including its perception of the environment such as captured image and lidar sensor output to the object level. At the object level, the agent processes perceptions to make a decision for the next step by using the RL Policy. For performance and success rate improvement, a pre-processing module is added before RL. However, adding this module has memory and power consumption overhead. In this work, we address this challenge in two ways: (1) by adding the meta-level to optimize the memory and power overhead of pre-processing module by switching between different models, and (2) by optimizing the YOLO model, called Squeezed Edge YOLO, to minimize the computational overhead. To do this, we consider multiple object detection models with different complexity.

Meta-level and object level detailed implementation. In this paper, we optimize two models: (1) Squeezed Edge YOLO and (2) CNN-based object detection model, however, the number of models can be extended to more than two models based on latency requirements of the application and the available memory constraint of the platform. Algorithm 1 gets parameters such as model output for the last two steps,

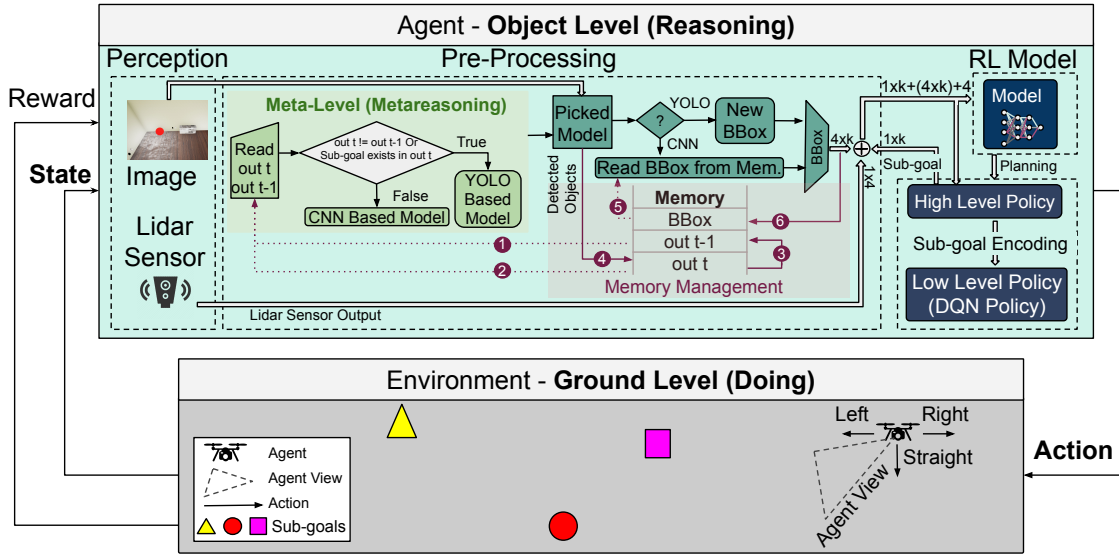


FIGURE 3. A system overview of the proposed approach which used metareasoning as a higher level to monitor the RL agent in real-time in an environment within multi goals. Metareasoning helps the agent to choose between multiple algorithms based on environmental changes to improve performance and energy efficiency.

Algorithm 1: Meta-Level Pseudo-Code

```

Data: squeezed_edge_YOLO_model,
        CNN_model, model_output_t,
        model_output_t - 1, sub_goal.
Result: picked_model.
picked_model  $\leftarrow$  null
if first step then
    picked_model  $\leftarrow$ 
        squeezed_edge_YOLO_model
end
if model_output_t  $\neq$  model_output_t - 1 OR
    sub_goal IN model_output_t then
    picked_model  $\leftarrow$ 
        squeezed_edge_YOLO_model
else
    picked_model  $\leftarrow$  CNN_model
end
return picked_model

```

$out_t - 1$ and out_t , and sub_goal given by the high level policy unit. It checks if out_t is as same as $out_t - 1$, then pick and returns the CNN-based model. But, if there is a change in object detection or the goal is within the detected object, it will pick Squeezed Edge YOLO to be executed for the next step. The meta-level will also pick Squeezed Edge YOLO for the first step in order to initialize the variables. Therefore, the object level is able to initialize out_t , $out_t - 1$

(put equal value for them), and BBox in memory. The object level executes the picked model by meta-level and updates out_t and $out_t - 1$ values. Then, use the new BBox if the YOLO-based model is executed, otherwise read BBox from memory. Finally, it concatenates BBox, Lidar sensor output, and sub-goal and passes a $1 \times k(4 \times k) + 4$ vector to RL.

Squeezed Edge YOLO. The goal is to deploy a compact variant of Edge YOLO on tiny low power edge devices, such as UAVs. As the microcontrollers on such edge devices are expected to perform a multitude of computationally intensive tasks, such as handling data from multiple input sources, digital signal processing, etc., modern microcontrollers often have relatively high clock speeds. Moreover, it is not uncommon to have hardware accelerators that can allow them to punch well above their form factor in certain operations. More often than not, the bottleneck in such edge devices is their memory capacity rather than their computational performance, with tiny edge devices typically being limited to a few megabytes, or less, of RAM. Due to transistor and energy budgets, it is not uncommon for microcontrollers to forego a hardware-managed cache, instead relying entirely on a software-managed memory hierarchy. This means that any software, including neural networks, has to consider its RAM usage and data locality if they want to extract the maximum amount of performance possible. As the first step for Edge YOLO optimizing, we

TABLE 1. Layer breakdown of the backbone feature extraction and the neck feature enhancement in Squeezed Edge YOLO

Backbone feature extraction				
Layer	Type	Size/Stride	Filters	Output
0	Conv	3x3/2	16	64x64x16
1	Conv	3x3/2	32	32x32x32
2	SE	-	-	32x32x32
3	Conv	3x3/1	16	32x32x16
4	SE	-	-	32x32x16
5	Conv	3x3/1	16	32x32x16
6	SE	-	-	32x32x16
7	Route	Input: [6, 4]	-	32x32x32
8	Conv	3x3/1	32	32x32x32
9	SE	-	-	32x32x32
10	Route	Input: [2, 9]	-	32x32x64
11	SE	-	-	32x32x64
12	MaxPool	2x2/2	-	16x16x64

Neck feature enhancement				
Layer	Type	Size/Stride	Filters	Output
0	Conv	3x3/1	64	16x16x64
1	Conv	3x3/2	128	8x8x128
2	SE	-	-	8x8x128
3	Conv	1x1/1	256	8x8x256
4	SE	-	-	8x8x256
5	Conv	1x1/1	256	8x8x256
6	Conv	3x3/1	128	8x8x128
7	Route	Input: [5,]	-	8x8x256
8	Conv	3x3/1	64	8x8x128
9	Upsample	2x2	-	16x16x128
10	Conv	1x1/1	64	16x16x64
11	Route	Input: [10, 0]	-	16x16x128
12	Conv	3x3/1	128	16x16x128
13	SE	-	-	16x16x128
14	Conv	1x1/1	128	16x16x128
15	SE	-	-	16x16x128
16	Conv	3x3/1	64	16x16x64
17	Detect	Input: [6, 16]	-	-

shrink the input image size from Edge YOLO's original 416x416 to 128x128. We did this because first, the target edge device, the AI-deck, can only capture images at a resolution of 324x244, meaning that the 416x416 input image size would end up partly redundant. Secondly, a 3-channel 128x128 image can fit entirely within the GAP8's L1 memory as it only occupies 49KB. We then began making adjustments to the remaining memory-heavy layers.

The most common culprits of high memory usage were large filter counts on the various convolutional layers, deeply nested skip connections, and 3x3 convolutional layers, as they require nine times the weights compared to 1x1 convolutional layers. This becomes a problem if the current layer inputs already occupy a large portion of memory. Besides iteratively making adjustments to the aforementioned items to bring their memory usage, and thus latency within reasonable limits, we also made several other major changes: (1) We

removed one of the detection heads, which allowed us to free up memory by shrinking the skip connection depth. (2) We removed the spectral pyramid pooling layer [11]. As one of its primary purposes is to enable arbitrary image input sizes, it was made partly redundant by the fact that we were using fixed images of 128x128 dimensions. Additionally, it was extremely taxing on the memory hierarchy, due to its size. (3) We implemented Squeeze and Excitation (SE) blocks in order to partly make up for the cut and slimmed layers. SE blocks are able to learn, and thus, enhance, inter-channel dependencies while still being computationally very lightweight. We thus added various convolutional layers as shown in Table 1.

Experimental Results

Experimental Setup

Dataset. We created two datasets for training and deployment, one for the purposes of person detection, and another composed of colored shapes for hierarchical navigation. The person dataset consists of approximately 3000 images captured from the AI-Deck camera in an 85/15 training/validation split. The shapes dataset contains 3500 images captured from both within the AirSim environment, as well as real world images, in a 90/10 training/validation split.

RL Training. To train the UAV agent, we used the Airsim environment, which is an open-source, cross-platform simulation environment developed by Microsoft that provides a realistic 3D environment for testing and developing autonomous vehicle systems. The primary objective of the RL agent in this context is to navigate through a sequence of goals in a specific predefined order. However, the order can be changed since the high-level controller allows the UAV to make autonomous decisions on how to tackle the sub-goals of a sequence. These decisions are guided by predefined conditions, such as reaching a certain number of steps or meeting specific threshold values for bounding box values, which influence the UAV's path and goal achievement. Unlike single-goal navigation, where the policy is usually a function that outputs actions based on a state, in multi-goal navigation, the policy function requires both a state and a goal to move between multiple goal locations. In addition to that, our reward function is designed to encourage reaching goals while avoiding collisions and uncompleted tasks.

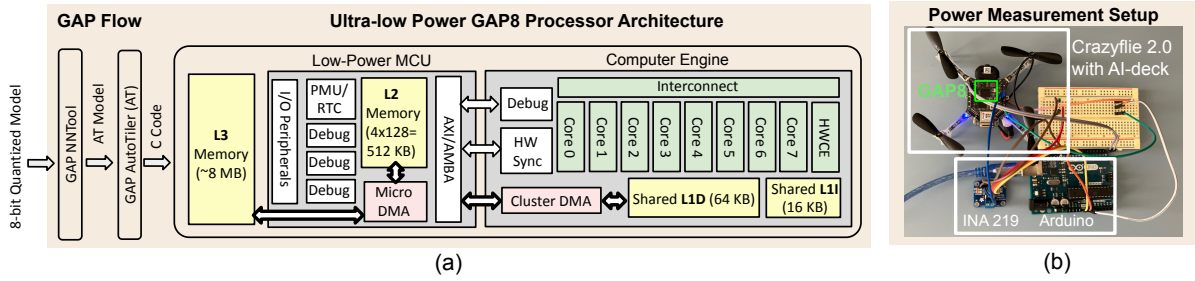


FIGURE 4. Gap8 processor architecture and power measurement setup for GAP8. (a) This Unit gets the 8-bit quantized object detection and reinforcement learning models and produces C code for deployment on embedded systems. (b) INA 219 and Arduino are used in this paper to measure the GAP8 processor power consumption while deploying the model onboard.

TABLE 2. For a fair comparison, we re-implement models and extract model size, power consumption, and inference latency of YOLOv5s [7], Edge YOLO [4], and Squeezed Edge YOLO proposed in this work. We successfully deployed Squeezed Edge YOLO on GAP8 processor, however, other models failed to be executed. Squeezed Edge YOLO is about 8x smaller than YOLOv5s.

Model	Parameters	Accuracy	Model Size	Power	Inference Latency
YOLOv5s [7]	7.3 M	0.96 mAP	237 Mb (32-bit)	N/A	N/A
Edge YOLO [4]	8.1 M	0.99 mAP	65 Mb (8-bit)	Failed due to memory limitations	
Squeezed Edge YOLO [This Work]	931 K (~8x smaller)	0.95 mAP	7.5 Mb (8-bit)	541 mW	130 ms

$$R_t = \begin{cases} R_{goal} & d_t \leq d_p \\ R_{coll} & d_o \leq d_q \\ R_{fail} & step = max_step \\ 0 & \end{cases} \quad (1)$$

where, R_{goal} , R_{coll} , and R_{fail} represent rewards for reaching the target, avoiding collisions, and coping with goal failure, respectively. The agent distances d_t and d_o are compared to predetermined thresholds d_p and d_q to determine these events. The details of training and the architecture are discussed in [7].

Crazyflie and GAP8 Processor. Figure 4 depicts the AI-deck that we targeted for deployment, which is an expansion board for the Crazyflie series of drones [12]. The AI-deck contains a RISC-V MCU named GAP8, composed of a primary core known as the Fabric Controller, and 8 cluster cores, as well as several hardware accelerators including integer vector units, and a convolution unit capable of 3x3, 5x5, and 4x7 convolutions. Despite its ambitious use case, with it being intended primarily for AI on the edge, several trade-offs were made in order to fit within its limited power budget, cost, and form factor - the primary ones being a lack of hardware floating point support, and a software-managed memory hierarchy, containing less than a megabyte of memory-mapped RAM. This software-managed memory hierarchy is composed of 3 layers: 64KB of fast, single-cycle L1 RAM, 512KB of slower, L2 RAM, and finally, a very slow off-chip non

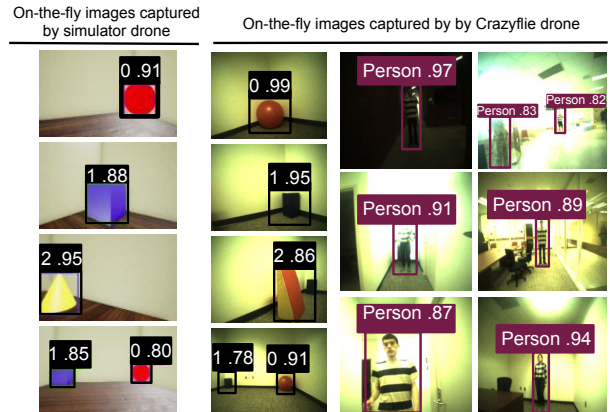


FIGURE 5. Squeezed edge YOLO object and human detection on Airsim simulation images (first column) and captured images by the RGB camera of Crazyflie in the real world (last three columns). The numbers show the classification and detected accuracy by squeezed edge YOLO model.

memory-mapped 8MB L3 RAM and 64MB L3 flash.

Model Optimization and Hardware Onboard Deployment Results

YOLO Model Optimization. Table 2 shows model details of YOLOv5s used in [7], Edge YOLO [4], and Squeezed Edge YOLO which is proposed in this work. The optimized resource-aware Squeezed Edge YOLO is a tinyML model about 8x smaller than the Edge

TABLE 3. (a) Counting how many times the meta-level picked the CNN-based model or YOLO-based model. We consider three different scenarios; the agent is trained to find one goal, two goals, and three goals in the environment. The YOLO-based model is prevented from being executed in the MetaE2RL approach at 36% on average which leads to energy and performance improvement. (b) The effect of considering metareasoning approach in pre-processing module. We compare model size, throughput, and energy per inference of the two proposed pre-processing models in this work which CNN model is consuming 42% energy per inference less. To calculate throughput and energy per inference, we measure power consumption and inference latency extracted from GAP8 processor when models are deployed onboard.

(a) MetaE2RL performance in Airsim environment

Number of Goals	Number of Picked Model		Success Rate
	CNN	Squeezed Edge YOLO	MetaE2RL
One Goal	9.1 K	7.8 K	100%
Two Goals	33 K	23 k	85%
Three Goals	30 K	108 K	80%

(b) Squeezed Edge YOLO and CNN model trade-off

Model	Parameters	Accuracy	Model Size	Throughput (Inference/Sec.)	Energy/Inference
Squeezed Edge YOLO	931 K	99%	7.5 Mb (8-bit)	8	70 mJ
CNN Model	430 K	98%	3.5 Mb (8-bit)	13 (1.6x faster)	40 mJ (42% more efficient)

YOLO and can be deployed on tiny edge devices. However, the previously proposed YOLO models cannot be deployed on tiny edge devices such as the GAP8 processor due to memory limitations. Figure 5 illustrates the Squeezed Edge YOLO test phase result on two shape and human detection datasets which could successfully detect objects on both simulation and real-world images.

MetaE2RL Airsim Simulation Result. MetaE2RL approach involves utilizing the DQN policy with carefully matched hyperparameters to the approach presented in ReProHRL [7]. Table 3 (a) presents the performance evaluation of the MetaE2RL approach in terms of achieving goals within a specified number of steps. For a single object, our policy achieved 100% performance within 17 K steps, during which the CNN model was executed for about 9 K steps. Additionally, MetaE2RL achieved 85% and 80% performance for two and three object detection, respectively. The YOLO-based model is prevented from being executed in the MetaE2RL approach at 36% on average which leads to energy and performance improvement.

Power and Latency Experiments. To show the performance and energy-efficiency of MetaE2RL, we measured the power and latency when deploying the Squeezed Edge YOLO and CNN model on the GAP8 processor's 8 RISC-V cluster cores. Table 3 (b) shows energy per inference for Squeezed Edge YOLO and the CNN model is 70 mJ and 40 mJ, respectively. The lighter CNN-based model, is 42% more energy efficient than Squeezed Edge YOLO. Moreover, with throughput equal to 13, it is 1.6x faster.

In order to show how energy efficiency would vary with different levels of model complexities, we extracted VCD hardware utilization traces. Figure 6 illustrates

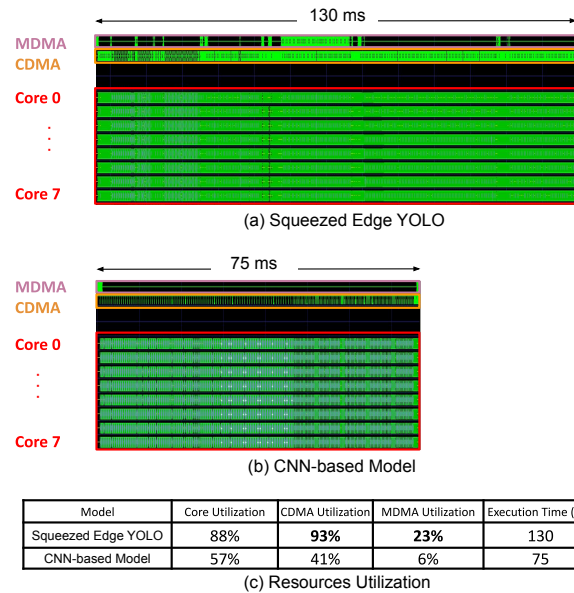


FIGURE 6. The VCD trace during the inference phase of onboard deployment on the GAP8 processor: (a) Squeezed Edge YOLO, (b) CNN-based model. The green portion represents hardware utilization. Cluster-DMA (CDMA) and Micro-DMA (MDMA) are responsible for data transfer between L1, L2 and L3 memories in GAP8 (as shown in Figure 4), respectively.

VCD traces including the Cluster-DMA (CDMA), Micro-DMA (MDMA), and core utilization for the CNN model and Squeezed Edge YOLO. As shown in Figure 4, the CDMA engine is responsible for moving data between L1 and L2 memory, L1 memory being an application-managed scratchpad. The MDMA is responsible for

transferring data between the different peripherals attached to the GAP8. For DNNs, it is used to move weights from what is known as L3 memory, essentially extra RAM attached to the HyperBus. This RAM is much slower, is not mapped into the GAP8's memory space, and is more power-intensive than L1 and L2 memory. However, it stores significantly more data [?]. In Figure 6 (a), the MDMA and CDMA units are both active nearly the entire inference phase, thus causing the processor to have to waste cycles waiting for data. Meanwhile, results for the lighter model, Figure 6 (b), show that MDMA and CDMA accesses are much less frequent for both DMA units. Moreover, near the end of the inference, the MDMA unit stops being used entirely, while the CDMA unit is accessed much less frequently, leading to lower energy consumption.

From additional experiments we performed, we found the "break-even" points in which data transfer costs began impacting latency and energy consumption primarily occurred when 1, layers were especially large. This would cause layers to require more transfers between L1 and L2, as well as possibly even L3 memory mid-layer-execution, causing stalls in cluster core execution. And 2, if the size of all the model parameters exceeds the overall L3 RAM memory.

Scalability of the Proposed Method: MetaE2RL.

For the GAP8 architecture, considering that we prefer to keep all models residing in the L3 RAM, we require the combined model size does not exceed the L3 RAM size of 8 MB. Additionally, when increasing the number of models, the overhead due to switching between more than two models can become significant if there is a need to repeatedly transfer data from L3, L2, and L1 RAM. Together these will impact the overall execution latency which shouldn't go below 8 frames per second or higher than 130 ms. For further investigation, we created several new models with 2, 4, and 8x larger numbers of parameters, and measured their latencies in the GAP8 GVSOC simulator. From our findings, models with large layer sizes, and entire networks with more parameters than available L3 RAM, increased the latency until it could no longer meet our real-time latency constraint of roughly 8 inferences a second.

Real-World Implementation of Baseline ReProHRL. In Figure 7, we implemented the baseline ReProHRL architecture [7] with real environments similar to Airsim and miniworld to validate sim2real transfer of these models. We considered Crazyflie as the UAV agent and JetBot with Nvidia Jetson Nano 4GB as the UGV agent in these environments. The RL policy for object navigation was successfully deployed without the need for retraining the model. This is

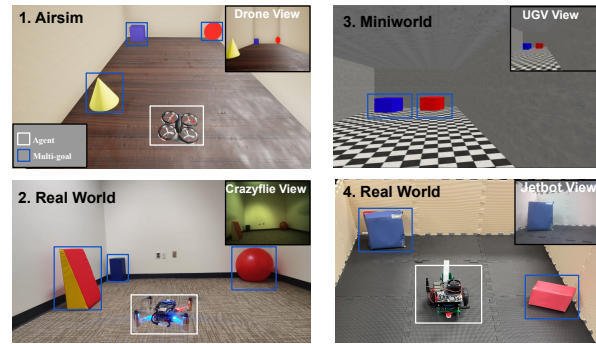


FIGURE 7. Upper row: Airsim and Miniworld simulation environments. Lower row: Crazyflie with AI-deck and Jetbot with Jetson Nano 4GB running the baseline architecture [7] in action in the real-world environment.

possible because our object detection model is trained using a combination of simulated and real-world images. Despite successfully navigating to all objects, the Crazyflie's onboard energy was inadequate for multi-goal navigation, with the drone taking between 30 seconds to 1 minute to reach an object in a 5mx5m room. The Jetbot exhibited similar performance to the Crazyflie but needs additional fine-tuning for hierarchical multi-goal navigation.

Conclusion

This work introduces a metareasoning approach for efficient multi-goal Reinforcement Learning (RL) deployment on edge devices. It begins by proposing two object detection models: Squeezed Edge YOLO and Light CNN-based Object Detection. By adding a Meta-Level to the architecture, it monitors the agent and environment in real-time, providing reasoning options to the agent. The approach optimizes resource usage for enhanced goal-oriented RL through pre-processing and Metareasoning. Evaluation is conducted on a Crazyflie drone with an ultra-low power GAP8 processor, resulting 8x smaller YOLO model, Squeezed Edge YOLO, with consuming 541 mW of power and 130 ms latency.

Acknowledgment

This project was sponsored by the U.S. Army Research Laboratory under Cooperative Agreement Number W911NF2120076.

REFERENCES

1. S. Kumar, P. Tiwari, and M. Zymbler, "Internet of things is a revolutionary approach for future technology enhancement: a review," *Journal of Big data*, vol. 6, no. 1, pp. 1–21, 2019.
2. Z. Wang et al., "Hierarchical memory-constrained operator scheduling of neural architecture search networks," in *Proceedings of the 59th ACM/IEEE Design Automation Conference*, pp. 493–498, 2022.
3. M. Navardi et al., "Toward real-world implementation of deep reinforcement learning for vision-based autonomous drone navigation with mission," *2022 Robotics: Science and Systems (RSS), 3rd Workshop on Closing the Reality Gap in Sim2Real Transfer Robotics*, 2022.
4. S. Liang, et al., "Edge YOLO: Real-Time Intelligent Object Detection System Based on Edge-Cloud Cooperation in Autonomous Vehicles" *IEEE Transactions on Intelligent Transportation Systems*, 2022.
5. J. Caylor et al., "Metareasoning for multi-criteria decision making using complex information sources," in *Artificial Intelligence and Machine Learning for Multi-Domain Operations Applications IV*, vol. 12113. SPIE, pp. 305–313, 2022.
6. M. Navardi, T. Mohsein, "MLAE2: Metareasoning for latency-aware energy-efficient autonomous nanodrones", *The IEEE International Symposium on Circuits and Systems (ISCAS)*, 2023.
7. T. Manjunath et al., "ReProHRL: Towards Multi-Goal Navigation in the Real World using Hierarchical Agents", *On 37th AAAI Conference on Artificial Intelligence, The 1st Reinforcement Learning Ready for Production workshop*, 2023.
8. E. Carrillo et al., "Communication-aware multi-agent metareasoning for decentralized task allocation", *IEEE Access*, vol. 9, pp. 98712–98730, 2021.
9. T. Schaul, D. Horgan, K. Gregor, and D. Silver, "Universal value function approximators", *IEEE International conference on machine learning*, 1312–1320, 2015.
10. S. Ren, K. He, R. Girshick, and J. Sun, "Towards an interpretable hierarchical agent framework using semantic goals", *arXiv preprint arXiv:2210.08412*, 2022.
11. H. Kaiming, Z. Xiangyu, R. Shaoqing, and S. Jian, "Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition", *Springer International Publishing*, 346–361 2014.
12. M. Navardi et al., "An optimization framework for efficient vision-based autonomous drone navigation", *2022 IEEE 4th International Conference on Artificial Intelligence Circuits and Systems (AICAS)*, 304–307, 2022.

Mozhgan Navardi is currently a Ph.D. student at the

Department of Electrical and Computer Engineering, Whiting School of Engineering, Johns Hopkins University, MD, USA. She received the MSc degree in Computer Engineering from Sharif University of Technology, Tehran, Iran, in 2019. Her research interests include low power embedded systems design and lifetime reliability management.

Edward Humes is currently pursuing a master's in Computer Science at University of Maryland, Baltimore County. His interests include embedded systems and computer architecture

Tejaswini Manjunath received her bachelor's degree in Computer Science from PES Institute of Technology, Bangalore, India. She graduated from the University of Maryland, Baltimore County (UMBC), MD, USA, with a master's degree in Computer Science and Engineering. Her research interests are mainly focused on machine learning, deep reinforcement learning, and computer architecture.

Tinoosh Mohsenin is currently an Associate Professor with the Department of Electrical and Computer Engineering, and the Institute of Assured Autonomy at Johns Hopkins University. She is the Director of the Energy Efficient High Performance Computing Lab and has authored or co-authored over 130 peer-reviewed journal and conference publications. Her research focus is on designing energy efficient embedded processors for machine learning and signal processing, knowledge extraction techniques for autonomous systems, wearable smart health monitoring, and embedded big data computing.